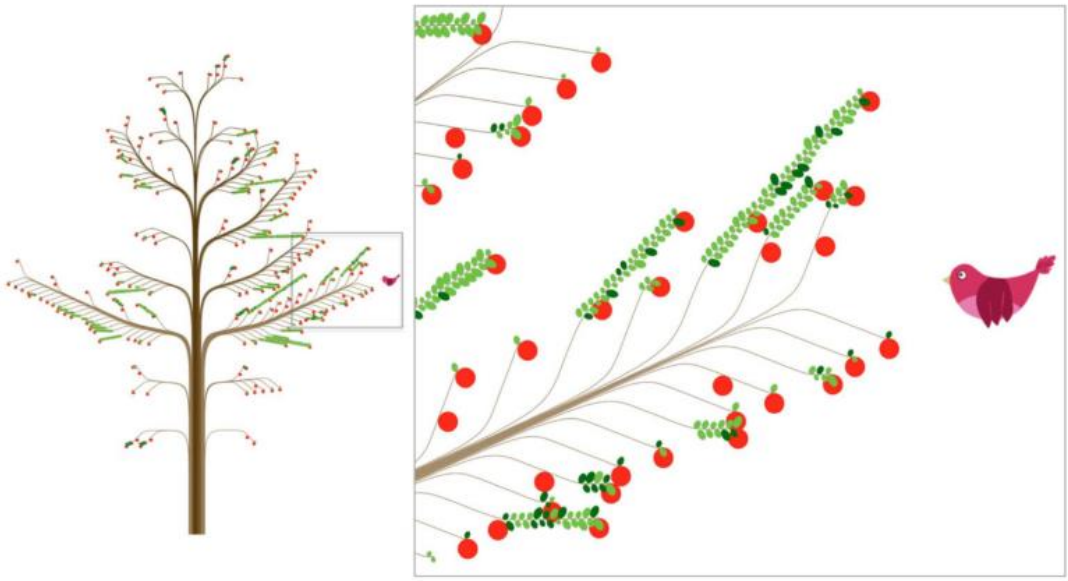
Weekly Report (2017.8.14-2017.8.20)

TASK	DEADLINE	CURRENT PROGRESS
Visual analysis system of LAN	8.31	Complete the development of a single tree structure. Now the problem is development of the relationship between the two trees. And current proposed solution is that clustering the leaf nodes by group before establishing the relationship between the two trees.

Done

1) Paper Reading: Contact Trees: Network Visualization beyond Nodes and Edges. The visualization design in this work is elegant. It is based on a botanical tree metaphor, thus it is easy to construct and the resulting tree-like visualization can display many properties at both tie and contact levels. The leaves and fruits on the tree symbolize interactions, contacts, or meetings among persons.



2) Learn how to implement the tree structure with the version 4 of D3.js. I got several usage of properties and functions that I have never used before. And I had a deeper understanding of what I already knew.

Related introduction of the tree:

https://github.com/d3/d3-hierarchy/blob/master/README.md#tree

In particular, I know more about the size() function of tree than before. As for radial tree, properties width and height represent the angel and radius respectively. This is corresponding to the normal tree.

- d3.stratify - create a new stratify operator.
- *stratify* - construct a root node from tabular data.
- *stratify*.id - set the node id accessor.
- *stratify*.parentId - set the parent node id accessor.

The ‘stratify’ is used in our system, transform the tabular data, such as CSV data, to hierarchical data to draw tree directly.

3) Explore the question: Is it faster to access data from file or database server?

> This is the kind of question that has no generic answer but is heavily dependent on th situation at hand. I even recently moved some data from a SQL database to a flat file system because the overhead of the DB, combined with some DB connection reliability issues, made using flat files a better choice.
>
> Some questions I would ask myself when making the choice include:
>
> 1. How am I consuming the data? For example will I just be reading from the beginning to the end rows in the order entered? Or will I be searching for rows that match multiple criteria?
> 2. How often will I be accessing the data during one program execution? Will I go once to get all books with Salinger as the author or will I go several times to get several different authors? Will I go more than once for several different criteria?
> 3. How will I be adding data? Can I just append a row to the end and that's perfect for my retrieval or will it need to be resorted?
> 4. *How logical will the code look in six months?* I emphasize this because I think this is too often forgotten in designing things (not just code, this hobby horse is actually from my days as a Navy mechanic cursing mechanical engineers). In six months when I have to maintain your code (or you do after working another project) which way of storing and retrieving data will make more sense. If going from flat files to a DB results in a 1% efficiency improvement but adds a week of figuring things out when you have to update the code have you really improved things.

From: https://stackoverflow.com/questions/2147902/is-it-faster-to-access-data-from-files-or-a-database-server

More answer about this questtion:

https://dba.stackexchange.com/questions/23124/whats-better-faster-mysql-or-filesystem

http://blog.csdn.net/jackpk/article/details/5954295

http://www.iteye.com/problems/30594

http://www.metsky.com/archives/313.html

At last, I determine to use file system and database server at the same time.

4) Make the leaf nodes invisible. Using filter() function.

> # *selection*.**filter**(*filter*) < >
>
> Filters the selection, returning a new selection that contains only the elements for which the specified *filter* is true. The *filter* may be specified either as a selector string or a function. If the *filter* is a function, it is evaluated for each selected element, in order, being passed the current datum (*d*), the current index (*i*), and the current group (*nodes*), with *this* as the current DOM element (*nodes*[*i*]).
>
> For example, to filter a selection of table rows to contain only even rows:
>
> ```
> var even = d3.selectAll("tr").filter(":nth-child(even)");
> ```
>
> This is approximately equivalent to using d3.selectAll directly, although the indexes may be different:
>
> ```
> var even = d3.selectAll("tr:nth-child(even)");
> ```

5) Using fast-csv npm module. Encounter the problem of encoding when read and write files. We can solve it by save the source file as a BOM-free utf-8 format.

6) Merge leaf nodes by changing the tree’s default separation() function, and modifying the symbol which will append to the node.

To Do

1) Finish the development of system as fast as I can.